Intelligent Autonomous Systems Project 3: SLAM

Ethan Glaser

eg492

I. Introduction and Problem Formulation

This project involves combining wheel encoding data and lidar measurement data to map a robot's environment as it navigates throughout a given space, while utilizing SLAM to handle noisy measurements by testing a number of options with similar trajectories and seeing which best matches up with what is expected.

II. Technical Approach

The first step involves reading in data for a specific case. This can be done using provided lidar and encoder reading functions based on the indicated data number. The encoding data is provided in 4 lists of encodings (one for each wheel) as well as a list of timestamps that correspond to the measurements. The lidar data is in dictionary format, with each containing a timestamp and a list of radian and distance pairs that correspond to a single reading. Once this data is read in, the lidar data is selected using the timestamps from the encodings – the closest timestamp in the lidar data is put into a new list for each timestamp from the encodings. Then lists of equivalent lengths of the timestamps, lidar readings, and all wheel encodings is returned. Additionally, the rear and front encodings are averaged to get a single list of values for the left and right side each.

For each timestamp in the data, the robot's position can be updated using algorithms discussed in lecture. Phi is initialized to 0 and then delta theta and delta x and y are determined using the equations below. Once this is complete, the global absolute position of the robot can be updated using phi and delta x and y, and then phi is updated using delta theta. The repetition of this process will lead to the plotting of the path of the robot as shown in Figure 2, and the tuning of the robot width parameter can be done in order to achieve a path that corresponds to the images provided for sample output.

$$\Delta\theta = \frac{e_R - e_L}{w} \qquad \Delta y = \frac{e_L + e_R}{2} \cos\frac{\Delta\theta}{2} \qquad \Delta x = \frac{e_L + e_R}{2} \sin\frac{\Delta\theta}{2}$$

Figure 1: Equations from lecture used to track the path of the robot

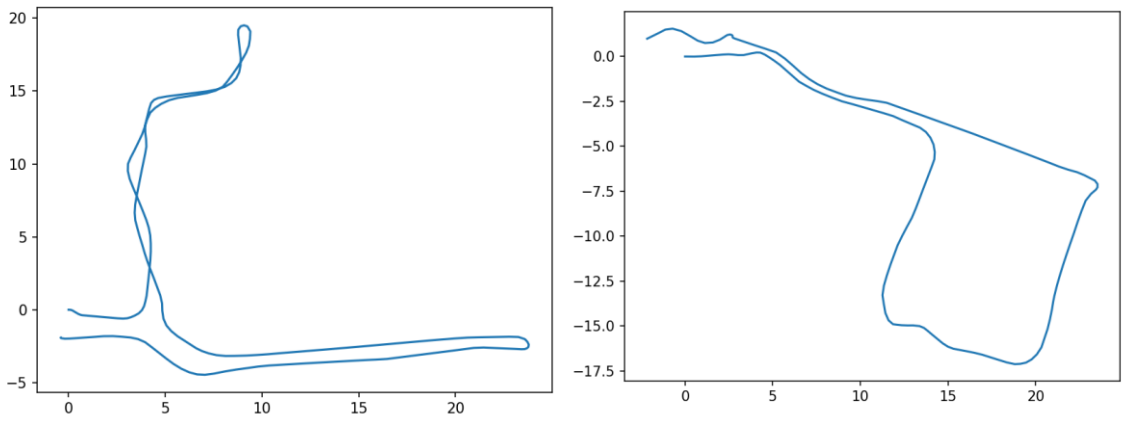

Figure 2: Plotted path of the robot in #20 and #23 after tuning the width hyperparameter

Once the location of the robot can be mapped, the next step is integrating the wall mapping using lidar data. At each timestamp, the robot's position and angle can be used to determine wall locations using the lidar distance and angles. After the robot position and wall positions are mapped onto the grid, a provided function can be used additionally to determine the location of gaps, or any grid locations between the robot and each wall reading where there is space. At every robot location, each lidar reading is used to find all locations where a wall is detected and space and this information is used to update the grid, subtracting if there is space and adding a constant if there is a wall. Once repeated across all the timestamps a map can be created.

The last step is adding in the SLAM component. This involves simulating multiple robots on the same data but with a small amount of noise added to the movements at each timestamp – specifically, the phi value. This helps with accounting for unexpected bumps or noise in the data. Only a single update is made from the many simulated robots – the one that most closely matches the current state of the map. At each time stamp, for each robot, the correlation between that robot's current view and the current state of the map are compared and a correlation coefficient is determined. These coefficients are used to adjust the weights of each robot over the duration of the time stamps. Then the weights of the robots are used to randomly select from the robots based on their weightings, and the selected robot is then used to update the map at that timestep. The weights fluctuate over the duration of the data due to the noise added into phi at each timestep, so various robots may be more likely to be selected at different points of time based on the random noise implemented.

There are many hyperparameters that were used in this algorithm and modifying these can lead to significant differences in the output in some cases. The first hyperparameter that was already mentioned is the robot width, which was visually determined using the path mapping function. There is also a hyperparameter for the number of robots used for the SLAM simulation. There is an "max_noise" hyperparameter which indicates the maximum amount of noise added to phi at each iteration. There is also an "update_threshold" parameter that allows for not updating the grid at a given timestep if the similarity between the walls at that timestep does not align closely enough with the map – which can be beneficial in avoiding adding updates that differ from the map but also makes detecting new walls harder. There is an additional hyperparameter indicating how much is subtracted from the grid index if a

gap is detected – the addition if a wall is detected is constant but this allows for an adjustment of the ratio between the two. There is also a “min_max_cap” that restricts how high an index value can reach, keeping the values in the grid within a certain range. Lastly there are parameters related to how finely the grid is handled as well as skipping timestamps for the purpose of speeding up testing.

```
for thresh in [-99, 0, 100, 300]:
    for angle in [0.001, 0.01, 0.1, 0.3]:
        for min_max in [5, 10, 25]:
            for sub in [0.4, 0.7, 1.3]:
                track_global_indices_SLAM(width, 45, 45, 90, 90, 3, filenum, 20, angle, thresh, sub, min_max_cap=min_max, averaging=20)
```

```
for filenum in [20, 23]:
    for thresh in [0]:#[-99, 0]:
        for angle in [0.01]:#[0.001, 0.01]:
            for min_max in [1,3,5,7]: # check smaller?
                for sub in [0.2,0.3,0.4,0.5]: # check smaller?
                    track_global_indices_SLAM(width, 45, 45, 90, 90, 3, filenum, 20, angle, thresh, sub, min_max_cap=min_max, averaging=20)
# import os
```

Figure 3: Iterations of hyperparameter tuning

III. Results and Discussion

The final results after hyperparameter tuning for all train and test maps are shown below. Overall, the odometry only performance was as expected, with a clear path displayed for the most part followed by some noise and misalignment of paths from the beginning to the end of the simulation. The implementation of SLAM and having multiple robots, which is meant to address this issue, was not as successful as I would have hoped, with the below images still deviating from the actual map as the timestamps draw near the end of the data. There are a few reasons why the SLAM implementation may not have worked as desired.

One potential reason is that the way correlation is being determined is not ideal. I am calculating correlation between a robot’s readings at a certain timestamp by summing the log loss values for the current map at any location where a wall is detected in that given timestamp. This can lead to a wide variety of results (even negative) that grow larger in magnitude as timestamp increases. The weights are then updated by multiplying the current weight by the current correlation and then normalizing the weights to add to 1. I fairly confident that something related to this process is not the proper way to do it I just was not sure how to calculate correlation and update the weights using the slides. Additionally, the noise values added to phi are relatively small because larger values lead to maps that go out of control. This would be something to adjust after the SLAM portion is fixed. Overall the below results from the SLAM implementation are more reflective of odometry only due to issues with the algorithmic approach to the SLAM portion of this project on my end.

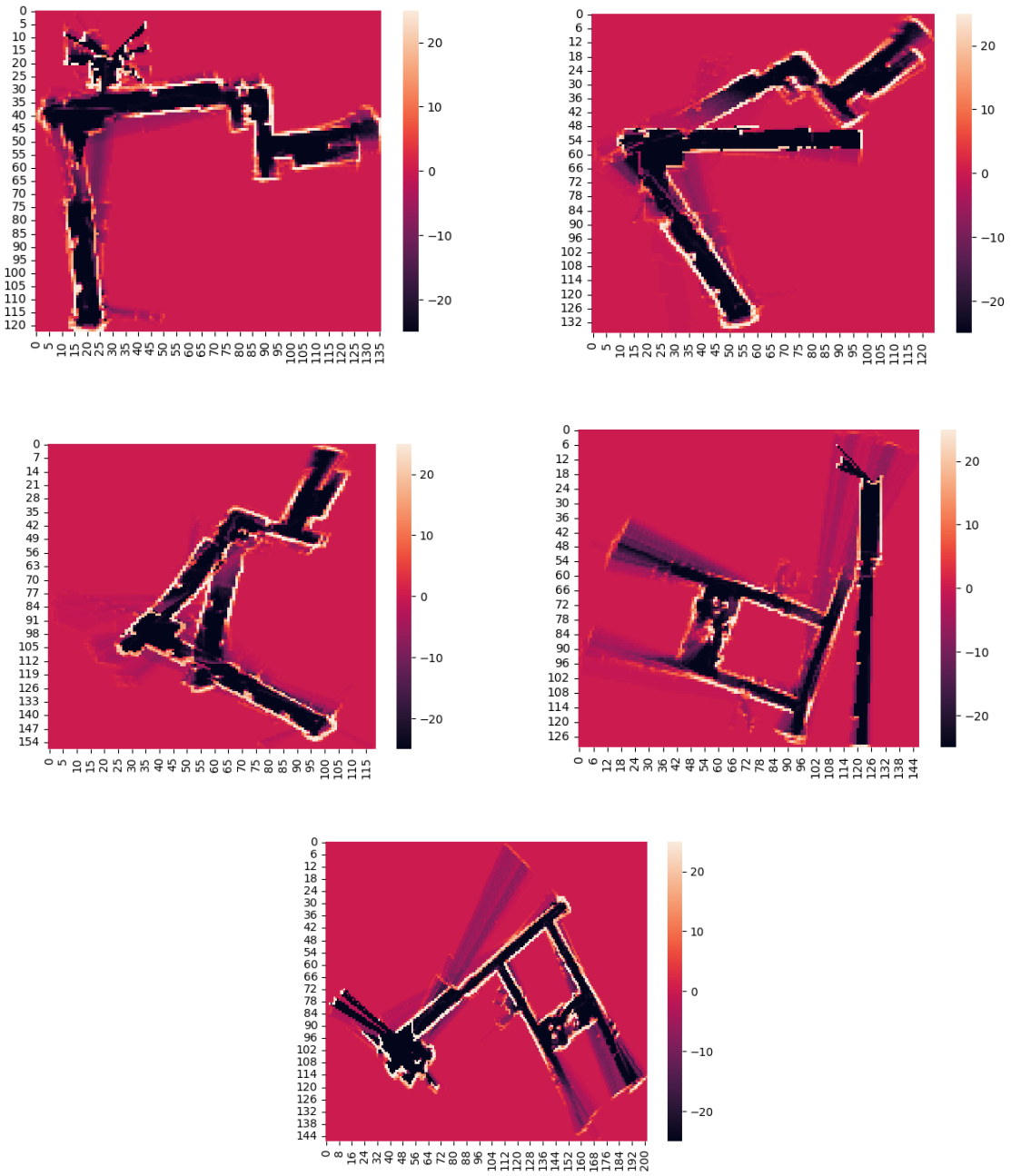

Figure 4: Results on tuned hyperparameters for 20, 21, 22, 23, 24